Distance to Circles in 3D

David Eberly, Geometric Tools, Redmond WA 98052
https://www.geometrictools.com/

Created: May 31, 2015
Last Modified: September 19, 2021

Contents

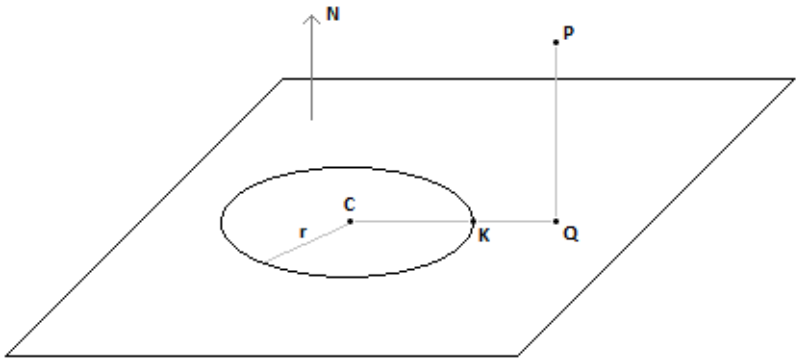
1 Introduction

A circle in 3D has a center $\boldsymbol{C}$, a radius r, and lies in a plane that contains the center and has unit-length normal $\boldsymbol{N}$. The circle is represented implicitly by $|\boldsymbol{X}-\boldsymbol{C}|^2 = r^2$ and $\boldsymbol{N}\cdot(\boldsymbol{X}-\boldsymbol{C}) = 0$. We wish to compute the distance between various objects and the circle. In this document, the specific objects are points, lines, and circles. The latter two objects are special cases of computing the distance from a parameterized curve to a circle.

2 Distance Between a Point and a Circle

A geometric argument allows us to determine the closest point in a coordinate-free manner. Figure 1 illustrates this.

Figure 1. The typical case for the circle point $\boldsymbol{K}$ closest to a specified point $\boldsymbol{P}$.



Define $\boldsymbol{\Delta} = \boldsymbol{P}-\boldsymbol{C}$. Let $\boldsymbol{Q}$ be the projection of $\boldsymbol{P}$ onto the plane of the circle; then

$$\boldsymbol{Q}-\boldsymbol{C} = \boldsymbol{\Delta} - (\boldsymbol{N}\cdot\boldsymbol{\Delta})\boldsymbol{N} \tag{1}$$

The circle point $\boldsymbol{K}$ closest to $\boldsymbol{P}$ is the projection of $\boldsymbol{Q}-\boldsymbol{C}$ onto the circle,

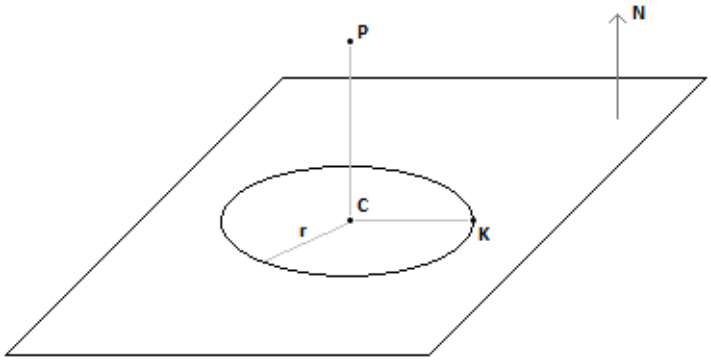
$$\boldsymbol{K} = \boldsymbol{C} + r\,(\boldsymbol{Q}-\boldsymbol{C})/|\boldsymbol{Q}-\boldsymbol{C}| = \boldsymbol{C} + r\,(\boldsymbol{\Delta} - (\boldsymbol{N}\cdot\boldsymbol{\Delta})\boldsymbol{N})/|\boldsymbol{\Delta} - (\boldsymbol{N}\cdot\boldsymbol{\Delta})\boldsymbol{N}| \tag{2}$$

The points $\boldsymbol{P}$, $\boldsymbol{Q}$, and $\boldsymbol{K}$ form a right triangle from which we can compute the squared distance between point and circle,

$$\begin{aligned} |\boldsymbol{P}-\boldsymbol{K}|^2 &= |\boldsymbol{P}-\boldsymbol{Q}|^2 + |\boldsymbol{Q}-\boldsymbol{K}|^2 \\ &= (\boldsymbol{N}\cdot\boldsymbol{\Delta})^2 + (|\boldsymbol{Q}-\boldsymbol{C}| - r)^2 \\ &= (\boldsymbol{N}\cdot\boldsymbol{\Delta})^2 + \left(\sqrt{|\boldsymbol{\Delta}|^2 - (\boldsymbol{N}\cdot\boldsymbol{\Delta})^2} - r\right)^2 \\ &= (\boldsymbol{N}\cdot\boldsymbol{\Delta})^2 + (|\boldsymbol{N}\times\boldsymbol{\Delta}| - r)^2 \end{aligned} \tag{3}$$

If the projection of $\boldsymbol{P}$ is exactly the circle center $\boldsymbol{C}$, then all points on the circle are equidistant from $\boldsymbol{C}$, as shown in figure 2.

Figure 2. All circle points are equidistant from $\boldsymbol{P}$.



The common squared distance is $|\boldsymbol{P}-\boldsymbol{K}|^2 = |\boldsymbol{P}-\boldsymbol{C}|^2 + |\boldsymbol{K}-\boldsymbol{C}|^2 = |\boldsymbol{\Delta}|^2 + r^2$, where $\boldsymbol{K}$ is any point on the circle. The squared distance formula is consistent with equation (3), because $(\boldsymbol{N}\cdot\boldsymbol{\Delta})^2 = |\boldsymbol{\Delta}|^2$ and $\boldsymbol{N}\times\boldsymbol{\Delta} = \boldsymbol{0}$.

The previous discussion is theoretical, assuming real-valued arithmetic. Listing 1 is a direct implementation of the algorithm. The inputs are the point $\boldsymbol{P}$ and the circle. When there is a unique circle point $\boldsymbol{K}$ closest to $\boldsymbol{P}$, the outputs are $\boldsymbol{K}$, the distance $|\boldsymbol{P}-\boldsymbol{K}|$ and a Boolean flag equidistant set to false. When $\boldsymbol{P}$ is on the normal line $\boldsymbol{C}+t\boldsymbol{N}$, the Boolean flag equidistant is set to true, the distance is from $\boldsymbol{P}$ to the circle and output $\boldsymbol{K}$ is invalid.

Listing 1. Pseudocode for the direct implementation of the point-circle algorithm in 3D assuming real-valued arithmetic. The type Vector3<Real> is a 3-tuple of real numbers that has associated arithemtic operations. The squared distance is also computed.

```
struct Circle3<Real>
{
    Vector3<Real> center;  // C
    Vector3<Real> normal;  // N
    Real radius;           // r
};

void DistanceClosestPoint(Vector3<Real> point, Circle3<Real> circle,
    Real& distance, Real& sqrDistance, Vector3<Real>& closest, bool& equidistant)
{
    Vector3<Real> delta = point - circle.center;  // Δ = P - C
    Real dotND = Dot(circle.normal, delta);  // N · Δ
    Vector3<Real> QmC = delta - dotND * circle.normal;  // Q - C
    Real lengthQmC = Length(QmC);  // |Q - C|
    if (lengthQmC > 0)
    {
        // The closest circle point to P is unique.
        Real crossND = Cross(circle.normal, delta);  // N × Δ
        Real radial = Length(crossND) - circle.radius;  // |N × Δ| - r
        sqrDistance = dotND * dotND + radial * radial;  // sqrt((N · Δ)^2 + (|N × Δ| - r)^2)
        distance = sqrt(sqrDistance);
        closest = circle.center + circle.radius * (QmC / lengthQmC);  // K = C + r(Q - C)/|Q - C|
        equidistant = false;
    }
```

```
    else
    {
        // All circle points are equidistant from P.
        sqrDistance = Dot(delta, delta) + circle.radius * circle.radius;
        distance = sqrt(sqrDistance); // sqrt(|Δ|^2 + r^2)
        closest = invalid;
        equidistant = true;
    }
}
```

Computing with floating-point arithmetic, a direct implementation has problems when $\boldsymbol{P}$ is nearly on the normal line $\boldsymbol{C} + t\boldsymbol{N}$. Rounding errors can lead to an inaccurate closest point $\boldsymbol{K}$. As an example, consider the circle with $\boldsymbol{C} = (0, 0, 0)$, $\boldsymbol{N} = (0, 0, 1)$ and $r = 1$. Let $\boldsymbol{P} = (0, 0, 1)$. All circle points are equidistant from $\boldsymbol{P}$ with distance $\sqrt{2}$. Floating-point rounding errors are not a problem with this dataset. However, if a rigid motion is applied to the point and circle, the errors are a problem. Listing 2 shows the results using double for the floating-point type Real.

Listing 2. Execution of the pseudocode when Real is double and when all circle points are equidistant from the point $\boldsymbol{P}$. Rather than return an invalid closest point, one of the circle points is returned.

```
// Compute a vector perpendicular to v.
template <typename Real>
Vector3<Real> GetPerpendicularTo(Vector3<Real> const& v)
{
    Real cMax = 0;
    size_t iMax = 0;
    for (size_t i = 0; i < 3; ++i)
    {
        Real c = std::fabs(v[i]);
        if (c > cMax)
        {
            cMax = c;
            iMax = i;
        }
    }

    Vector3<Real> perpendicular{};
    if (cMax > 0.0)
    {
        size_t iNext = (iMax + 1) % 3;
        perpendicular[iMax] = v[iNext];
        perpendicular[iNext] = -v[iMax];
        Normalize(perpendicular);
    }
    return perpendicular;
}

template <typename Real>
void DistanceClosestPoint(Vector3<Real> point, Circle3<Real> circle,
    Real& distance, Real& sqrDistance, Vector3<Real>& closest, bool& equidistant)
{
    Vector3<Real> delta = point - circle.center;
    Real dotND = Dot(circle.normal, delta);
    Vector3<Real> QmC = delta - dotND * circle.normal;
    Real lengthQmC = Length(QmC);
    if (lengthQmC > 0)
    {
        // The closest circle point to P is unique.
        Real crossND = Cross(circle.normal, delta);
        Real radial = Length(crossND) - circle.radius;
        sqrDistance = dotND * dotND + radial * radial;
        distance = sqrt(sqrDistance);
        closest = circle.center + circle.radius * (QmC / lengthQmC);
```

```
        equidistant = false;
    }
    else
    {
        // All circle points are equidistant from P.
        sqrDistance = Dot(delta, delta) + circle.radius * circle.radius;
        distance = sqrt(sqrDistance);
        closest = circle.center + circle.radius * GetPerpendicularTo(circle.normal);
        equidistant = true;
    }
}

void main()
{
    Vector3<double> point{};
    Circle3<double> circle{};
    double distance{};
    Vector3<double> closest{};
    bool equidistant{};

    point = { 0.0, 0.0, 1.0 };
    circle.center = { 0.0, 0.0, 0.0 };
    circle.normal = { 0.0, 0.0, 1.0 };
    circle.radius = 1.0;
    DistanceClosestPoint(point, circle, distance, closest, equidistant);
    // distance = 1.4142135623730947 (approximately √2)
    // sqrDistance = 2.0
    // closest = { -1.0, 0.0, 0.0 }
    // equidistant = true

    // Generate a random rigid motion.
    Vector3<double> translate{ 0.1234, 5.6789, -1.9735 };
    Matrix3x3<double> rotate{};  // Matrix stored in row-major order.
    Quaternion<double> q(1.0, 2.0, 3.0, 4.0);
    Normalize(q);
    Convert(q, rotate); // Convert a quaternion to a rotation matrix.
    // With exact arithmetic, rotate = {{4,−20,22},{28,10,4},{−10,20,20}}/30
    // With double arithmetic,
    // rotate =
    // {
    //  {0.13333333333333353,-0.66666666666666663,0.73333333333333317}
    //  {0.93333333333333324,0.33333333333333348,0.13333333333333336}
    //  {-0.33333333333333326,0.66666666666666652,0.66666666666666674}
    // }
    // The matrix entries each have a small rounding error.

    point = rotate * point + translate;
    // {0.85673333333333312,5.8122333333333334,-1.3068333333333333}
    circle.center = rotate * circle.center + translate;
    // {0.12340000000000000,5.6788999999999996,-1.9735000000000000}
    circle.normal = rotate * circle.normal;
    // {0.73333333333333317,0.13333333333333336,0.66666666666666674}
    DistanceClosestPoint(point, circle, distance, closest, equidistant);
    // distance = 1.4142135623730947 (approximately √2)
    // sqrDistance = 1.9999999999999991
    // closest = { 0.37291314462072211, 6.6145742923277080, -1.7239868553792779 }
    // equidistant = false

    Vector3<double> diff = closest - circle.center;
    double dotNDiff = Dot(circle.normal, diff); // 0.47407497477937210
    double lengthDiff = Length(diff); // 1.0000000000000002
}
```

In theory, $\boldsymbol{K}$ is on the circle, which means $\boldsymbol{N} \cdot (\boldsymbol{K} - \boldsymbol{C}) = 0$ and $|\boldsymbol{K} - \boldsymbol{C}| = r$. The value of dotNDiff

should be zero, but rounding errors have caused it to be significantly different from zero. The length of QmC is nearly zero, $4.4495572620543707 \times 10^{-16}$ which is effectively noise. The normalization QmC/lengthQmC greatly magnifies the noise, leading to the inaccurate result.

Experiments have shown that the rounding-error behavior is better by computing an orthonormal basis $\{\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{N}\}$, where the vectors are unit length and mutually perpendicular. The difference $\boldsymbol{\Delta}$ is

$$\boldsymbol{\Delta} = (\boldsymbol{U} \cdot \boldsymbol{\Delta})\boldsymbol{U} + (\boldsymbol{V} \cdot \boldsymbol{\Delta})\boldsymbol{V} + (\boldsymbol{N} \cdot \boldsymbol{\Delta})\boldsymbol{N} \tag{4}$$

The projection of $\boldsymbol{Q} - \boldsymbol{C}$ is

$$\boldsymbol{Q} - \boldsymbol{C} = (\boldsymbol{U} \cdot \boldsymbol{\Delta})\boldsymbol{U} + (\boldsymbol{V} \cdot \boldsymbol{\Delta})\boldsymbol{V} \tag{5}$$

The computation of $\boldsymbol{U}$ and $\boldsymbol{V}$ involves normalizations (divisions by square roots) which can be avoided by instead computing an *orthogonal* basis $\{\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{N}\}$, where the vectors are mutually perpendicular but not required to be unit length. They are computed so that $\boldsymbol{V} = \boldsymbol{N} \times \boldsymbol{U}$ with $|\boldsymbol{V}| = |\boldsymbol{N}||\boldsymbol{U}|$. The projection of $\boldsymbol{Q} - \boldsymbol{C}$ is

$$\boldsymbol{Q} - \boldsymbol{C} = \left(\frac{\boldsymbol{U} \cdot \boldsymbol{\Delta}}{\boldsymbol{U} \cdot \boldsymbol{U}}\right)\boldsymbol{U} + \left(\frac{\boldsymbol{V} \cdot \boldsymbol{\Delta}}{\boldsymbol{V} \cdot \boldsymbol{V}}\right)\boldsymbol{V} = \left(\frac{\boldsymbol{U} \cdot \boldsymbol{\Delta}}{\boldsymbol{U} \cdot \boldsymbol{U}}\right)\boldsymbol{U} + \left(\frac{\boldsymbol{V} \cdot \boldsymbol{\Delta}}{(\boldsymbol{N} \cdot \boldsymbol{N})(\boldsymbol{U} \cdot \boldsymbol{U})}\right)\boldsymbol{V} \tag{6}$$

It is sufficient to normalize the scaled vector

$$\boldsymbol{S} = (\boldsymbol{N} \cdot \boldsymbol{N})(\boldsymbol{U} \cdot \boldsymbol{U})(\boldsymbol{Q} - \boldsymbol{C}) = (\boldsymbol{N} \cdot \boldsymbol{N})(\boldsymbol{U} \cdot \boldsymbol{\Delta})\boldsymbol{U} + (\boldsymbol{V} \cdot \boldsymbol{\Delta})\boldsymbol{V} \tag{7}$$

which avoids the divisions and square roots associated with the normalizations in listing **??**. The Geometric Tools code for computing the orthogonal basis is shown in listing 3.

Listing 3. Code for computing an orthogonal basis from 1 or 2 linearly independent vectors. In the distance application, the number of inputs is 1.

```
// If numInputs is 1, v0 must be a user-provided nonzero vector.
// If numInputs is 2, v0 and v1 must be user-provided linearly independent vectors.
template <typename Real>
bool ComputeOrthogonalBasis(size_t numInputs, Vector3<Real>& v0, Vector3<Real>& v1, Vector3<Real>& v2)
{
    if (numInputs == 1)
    {
        if (std::fabs(v0[0]) > std::fabs(v0[1]))
        {
            v1 = { -v0[2], 0, v0[0] };
        }
        else
        {
            v1 = { 0, v0[2], -v0[1] };
        }
    }
    else // numInputs == 2 || numInputs == 3
    {
        v1 = Dot(v0, v0) * v1 - Dot(v1, v0) * v0;
    }

    if (v1 == Vector3<Real>::Zero())
    {
        v2.MakeZero();
        return false;
    }

    v2 = Cross(v0, v1);
    return v2 != Vector3<Real>::Zero();
}

// For the distance query:
Vector3<Real> U{}, V{}, N = circle.normal;
ComputeOrthogonalBasis(1, N, U, V);
```

Listing 1 is modified to a more robust one shown in listing 4.

Listing 4. Pseudocode for a robust implementation of the point-circle algorithm in 3D assuming floating-point arithmetic.

```
template <typename Real>
void DistanceClosestPoint(Vector3<Real> point, Circle3<Real> circle,
    Real& distance, Real& sqrDistance, Vector3<Real>& closest, bool& equidistant)
{
    Vector3<Real> PmC = point - circle.center;
    Vector3<Real> U{}, V{}, N = circle.normal;
    ComputeOrthogonalBasis(1, N, U, V);
    Vector3<Real> scaledQmC = (Dot(N, N) * Dot(U, PmC)) * U + Dot(V, PmC) * V;
    Real lengthScaledQmC = Length(scaledQmC);
    if (lengthScaledQmC > 0)
    {
        // The closest circle point to P is unique.
        closest = circle.center + circle.radius * (scaledQmC / lengthScaledQmC);
        Real height = Dot(N, PmC);
        Real radial = Length(Cross(N, PmC)) - circle.radius;
        sqrDistance = height * height + radial * radial;
        distance = std::sqrt(sqrDistance);
        equidistant = false;
    }
    else
    {
        // All circle points are equidistant from P.
        closest = circle.center + circle.radius * GetOrthogonal(N, true);
        sqrDistance = Dot(PmC, PmC) + circle.radius * circle.radius;
        distance = std::sqrt(sqrDistance);
        equidistant = true;
    }
}
```

Function Result operator()(Vector3<Real> const& P, Circle3<Real> const& circle) is the implementation provided in the file DistPoint3Circle3.h.

3 Distance Between a Curve and a Circle

Equations (2) and (3) were derived for a single point $\boldsymbol{P}$. A curve provides a parameterized point $\boldsymbol{P}(t)$ for variable t. All occurrences in the equations of $\boldsymbol{P}$ and $\boldsymbol{K}$ can be replaced with time-varying values. The closest point on the circle to $\boldsymbol{P}(t)$ is

$$\boldsymbol{K}(t) = \boldsymbol{C} + r\,(\boldsymbol{\Delta}(t) - (\boldsymbol{N}\cdot\boldsymbol{\Delta}(t))\boldsymbol{N})/|\boldsymbol{\Delta}(t) - (\boldsymbol{N}\cdot\boldsymbol{\Delta}(t))\boldsymbol{N}| \tag{8}$$

and the half-squared-distance function is

$$F(t) = |\boldsymbol{P}(t) - \boldsymbol{K}(t)|^2/2 = ((\boldsymbol{N}\cdot\boldsymbol{\Delta}(t))^2 + (|\boldsymbol{N}\times\boldsymbol{\Delta}(t)| - r)^2)/2 \tag{9}$$

where $\boldsymbol{\Delta}(t) = \boldsymbol{P}(t) - \boldsymbol{C}$ and where the first equality defines the squared-distance function $F(t)$.

The set of points on the circle closest to the curve occur at the critical points of $F(t)$. These are points for which the derivative $F'(t)$ is zero or undefined. The derivative is

$$\begin{aligned} F'(t) &= (\boldsymbol{N}\cdot\boldsymbol{\Delta}(t))(\boldsymbol{N}\cdot\boldsymbol{\Delta}'(t)) + (|\boldsymbol{N}\times\boldsymbol{\Delta}(t)| - r)\,(\boldsymbol{N}\times\boldsymbol{\Delta}(t)\cdot\boldsymbol{N}\times\boldsymbol{\Delta}'(t))/|\boldsymbol{N}\times\boldsymbol{\Delta}(t)| \\ &= \boldsymbol{\Delta}(t)\cdot\boldsymbol{\Delta}'(t) - r\,(\boldsymbol{N}\times\boldsymbol{\Delta}(t)\cdot\boldsymbol{N}\times\boldsymbol{\Delta}'(t))/|\boldsymbol{N}\times\boldsymbol{\Delta}(t)| \end{aligned} \tag{10}$$

The derivative is undefined when $|\boldsymbol{N}\times\boldsymbol{\Delta}(t)| = |\boldsymbol{\Delta}(t) - (\boldsymbol{N}\cdot\boldsymbol{\Delta}(t))\boldsymbol{N}| = 0$. The two types of critical points are solutions to a single equation,

$$G(t) = |\boldsymbol{N}\times\boldsymbol{\Delta}(t)|\boldsymbol{\Delta}(t)\cdot\boldsymbol{\Delta}'(t) - r\boldsymbol{N}\times\boldsymbol{\Delta}(t)\cdot\boldsymbol{N}\times\boldsymbol{\Delta}'(t) = 0 \tag{11}$$

where the first equality defines the function $G(t)$.

The method of solving $F'(t) = 0$ depends on the nature of the curve $\boldsymbol{P}(t)$. A general root-bounding algorithm may be used; for each interval containing a single root, bisection may be used to estimate the root. $F(t)$ is evaluated for each critical value t, and the smallest value $\hat{F}$ is the distance from the curve to the circle. The minimum is attained by at least one $\hat{t}$, where by definition $F(\hat{t}) = \hat{F}$, but there may be multiple minimizers. For each minimizer $\hat{t}$, we can compute a closest point $\boldsymbol{K}(\hat{t})$.

For some curves, equation (11) may be manipulated to produce a polynomial equation. In particular, this is possible for lines and circles, each discussed in a later section. The idea is to rearrange the terms of the equation and square them to obtain

$$H(t) = |\boldsymbol{N}\times\boldsymbol{\Delta}(t)|^2(\boldsymbol{\Delta}(t)\cdot\boldsymbol{\Delta}'(t))^2 - r^2(\boldsymbol{N}\times\boldsymbol{\Delta}(t)\cdot\boldsymbol{N}\times\boldsymbol{\Delta}'(t))^2 = 0 \tag{12}$$

where the first equality defines the function $H(t)$. When $\boldsymbol{\Delta}(t)$ is a polynomial curve, $H(t)$ is a polynomial in t. The prototypical case is when $\boldsymbol{P}(t)$ is a line. The case of a circle (distance between two circles) generates H as a polynomial of $\cos(t)$ and $\sin(t)$. The term $\sin(t)$ can be eliminated using the resultant of $H(t)$ and $\sin(t)^2 + \cos(t)^2 - 1$ to produce a high-degree polynomial equation to solve. The squaring of terms will introduce extraneous roots, but the corresponding candidate points on the circle will be farther away than others and are discarded by the process. You may also compute a solution to $H(t) = 0$ and then substitute into $G(t)$. If the result is zero (or nearly zero when computing with floating-point arithmetic), the t-value is used to compute a candidate closest point.

4 Distance Between a Line and a Circle

Two algorithms are discussed here. The first algorithm is based on computing roots using equation (12), which turns out to be a polynomial equation of degree at most 4. The second algorithm computes roots of equation (10) using root bounding and bisection.

4.1 Polynomial-Based Algorithm

The line is parameterized by $\boldsymbol{P}(t) = \boldsymbol{B} + t\boldsymbol{M}$, where $\boldsymbol{M}$ is a unit-length vector and t is any real number. Define $\boldsymbol{D} = \boldsymbol{B} - \boldsymbol{C}$ so that $\boldsymbol{\Delta}(t) = \boldsymbol{D} + t\boldsymbol{M}$. Consequently, $\boldsymbol{\Delta}'(t) = \boldsymbol{M}$, $\boldsymbol{N}\times\boldsymbol{\Delta}(t) = \boldsymbol{N}\times\boldsymbol{D} + t\boldsymbol{N}\times\boldsymbol{M}$,

and $\boldsymbol{N} \times \boldsymbol{\Delta}'(t) = \boldsymbol{N} \times \boldsymbol{M}$. Equation (12) becomes

$$\begin{aligned} H(t) &= \left(|\boldsymbol{N} \times \boldsymbol{M}|^2 t^2 + 2\boldsymbol{N} \times \boldsymbol{M} \cdot \boldsymbol{N} \times \boldsymbol{D} t + |\boldsymbol{N} \times \boldsymbol{D}|^2\right)(t + \boldsymbol{M} \cdot \boldsymbol{D})^2 \\ &\quad -r^2\left(|\boldsymbol{N} \times \boldsymbol{M}|^2 t + \boldsymbol{N} \times \boldsymbol{M} \cdot \boldsymbol{N} \times \boldsymbol{D}\right)^2 \\ &= 0 \end{aligned} \tag{13}$$

The function $H(t)$ is a polynomial of degree 4. Compute its real-valued roots. For each root $\hat{t}$, evaluate $F(\hat{t})$ and keep track of the minimum F-value, $\hat{F}$. The circle points closest to the line are $\boldsymbol{K}(\hat{t})$ (when defined) for each root $\hat{t}$ that attains the minimum $\hat{F}$. It is also possible that a root corresponds to a point on the normal line $\boldsymbol{C} + t\boldsymbol{N}$, in which case all circle points are equidistant but you may as well choose one of them.

Several special geometrical cases are of interest.

1. If the line is perpendicular to the plane and contains the circle center, then $\boldsymbol{C}$ attains minimum distance (r) to the circle and all circle points are equidistant from the center. This case has $\boldsymbol{N} \times \boldsymbol{M} = \boldsymbol{0}$ and $\boldsymbol{N} \times \boldsymbol{D} = \boldsymbol{0}$, so $H(t)$ is identically zero.

2. If the line is perpendicular to the plane but does not contain the circle center, then the intersection of the line and plane is the line point closest to the circle. This case has $\boldsymbol{N} \times \boldsymbol{M} = \boldsymbol{0}$, $\boldsymbol{N} \times \boldsymbol{D} \neq \boldsymbol{0}$, and $H(t) = |\boldsymbol{N} \times \boldsymbol{D}|^2(t + \boldsymbol{M} \cdot \boldsymbol{D})^2$. The root to $H(t) = 0$ is $t = -\boldsymbol{M} \cdot \boldsymbol{D}$, which indeed is the parameter for the intersection of the line and the plane.

3. If the line is not perpendicular to the plane but contains the circle center, then $\boldsymbol{N} \times \boldsymbol{M} \neq \boldsymbol{0}$, $\boldsymbol{D} = \boldsymbol{0}$, and $H(t) = |\boldsymbol{N} \times \boldsymbol{M}|^2 t^4 - r^2|\boldsymbol{N} \times \boldsymbol{M}|^4 t^2$. The roots to $H(t) = 0$ are $t = 0$ and $t = \pm r|\boldsymbol{N} \times \boldsymbol{M}|$. The root $t = 0$ is not a minimizer, but the other two roots are—the minimum distance is attained by two line-circle point pairs.

4. If the line is parallel to the plane, then $\boldsymbol{N} \cdot \boldsymbol{M} = 0$ and $\{\boldsymbol{N}, \boldsymbol{M}, \boldsymbol{N} \times \boldsymbol{M}\}$ is a right-handed orthonormal set. The vector $\boldsymbol{N} \times \boldsymbol{D}$ is perpendicular to $\boldsymbol{N}$, so $\boldsymbol{N} \times \boldsymbol{D} = u\boldsymbol{M} + v\boldsymbol{N} \times \boldsymbol{M}$, where $u = -\boldsymbol{N} \times \boldsymbol{M} \cdot \boldsymbol{D}$ and $v = \boldsymbol{M} \cdot \boldsymbol{D}$. It follows that $H(t) = (t + v)^2[(t + v)^2 - (r^2 - u^2)]$. The equation has a real root of multiplicity 2 when $r^2 < u^2$, a real root of multiplicity 4 when $r^2 = u^2$, or three distinct real roots (one has multiplicity 2, the other two are simple) when $r^2 > u^2$. If you view the configuration projected onto the plane of the circle, the case $r^2 < u^2$ corresponds to the line outside the circle, so there is a single closest point generated by $t = -v$. The case $r^2 = u^2$ corresponds to the line tangent to the circle, so there is a single closest point generated by $t = -v$. The case $r^2 > u^2$ corresponds to the line intersecting the circle in two points which are generated by $t = -v \pm \sqrt{r^2 - u^2}$.

An implementation is provided in the file DistLine3Circle3.h; see the function

```
Result operator()(Line3<Real> const& line, Circle3<Real> const& circle)
```

4.2 Nonpolynomial-Based Algorithm

4.2.1 Canonical Form

Equation (10) appears complicated, but a reduction of the line and circle to a canonical form allows us to understand the geometry as it relates to the analysis of critical points. Specifically, choose the circle center

to be at the origin, $\boldsymbol{C} = (0, 0, 0)$, and choose the normal vector to be $\boldsymbol{N} = (0, 0, 1)$. We can accomplish this by a translation and rotation of the original circle, and this transformation can be applied to the line.

Line and Normal are Parallel

The special case when $\boldsymbol{M}$ is parallel to $(0, 0, 1)$ is easy to analyze. The point on the line closest to the circle is the intersection of the line with the plane. In the canonical form, the line is $(0, 0, m_2)t + (b_0, b_1, b_2)$ for $|m_2| = 1$. The intersection point is $(b_0, b_1, 0)$ and occurs when $t = -b_2/m_2$. If $(b_0, b_1) \neq (0, 0)$, the closest circle point is $r(b_0, b_1)/\sqrt{b_0^2 + b_1^2}$. If $(b_0, b_1) = (0, 0)$, all circle points are equidistant from the line point.

Line and Normal are not Parallel: Special Case

Consider when $\boldsymbol{M}$ is not parallel to $(0, 0, 1)$. The circle is invariant to a rotation about the normal vector, so we may apply such a rotation so that the line direction is $\boldsymbol{M} = (m_0, 0, m_2)$ with $m_0^2 + m_2^2 = 1$ and $m_0 > 0$. The line is guaranteed to intersect the x_1x_2-plane, so we may choose a point on the line $\boldsymbol{B} = (0, b_1, b_2)$.

Suppose that $b_1 = 0$. The line point is $\boldsymbol{B} = (0, 0, b_2)$ and is equidistant from the circle points. We know that $m_0 > 0$, so geometrically there must be points on the line closer to the circle than $(0, 0, b_2)$ is. In fact, the only closest-point candidates on the circle are $\pm(r, 0, 0)$. If we solve $\boldsymbol{M} \cdot (\boldsymbol{P}(t) \pm (r, 0, 0)) = 0$, we obtain $t = -m_2b_2 \pm rm_0$. The line point closest to $(r, 0, 0)$ is $\boldsymbol{P}_0 = (-m_2b_2 + rm_0)(m_0, 0, m_2) + (0, 0, b_2)$ and the line point closest to $(-r, 0, 0)$ is $\boldsymbol{P}_1 = (-m_2b_2 - rm_0)(m_0, 0, m_2) + (0, 0, b_2)$. Define $d_0 = |\boldsymbol{P}_0 - (r, 0, 0)|$ and $d_1 = |\boldsymbol{P}_1 - (-r, 0, 0)|$. Some algebra will show that $d_0^2 = (m_2r + m_0b_2)^2$, $d_1^2 = (m_2r - m_0b_2)^2$, and $d_0^2 - d_1^2 = 4rm_0m_2b_2$. Thus, $\text{Sign}(d_0^2 - d_1^2) = \text{Sign}(m_2b_2)$, which allows us to decide the closest points solely by analyzing the sign of m_2b_2. In the special case $m_2 = 0$, the line is parallel to the plane of the circle and we have a pair of closest points. In the special case $b_2 = 0$, the line passes through the circle center, again giving us a pair of closest points.

Line and Normal are not Parallel: General Case

The direction $\boldsymbol{M}$ is not parallel to $(0, 0, 1)$ but now we cnosider when $b_1 \neq 0$. Equation (10) reduces to

$$F'(t) = t + m_2b_2 - \frac{rm_0^2t}{\sqrt{m_0^2t^2 + b_1^2}} \tag{14}$$

We will work with $F'(t)$ directly to obtain a numerically robust algorithm for computing roots.

Define the function

$$G(t) = \frac{rm_0^2t}{(m_0^2t^2 + b_1^2)^{1/2}} \tag{15}$$

so that $F'(t) = t + m_2b_2 - G(t)$. The first derivative is

$$G'(t) = \frac{rm_0^2b_1^2}{(m_0^2t^2 + b_1^2)^{3/2}} \tag{16}$$

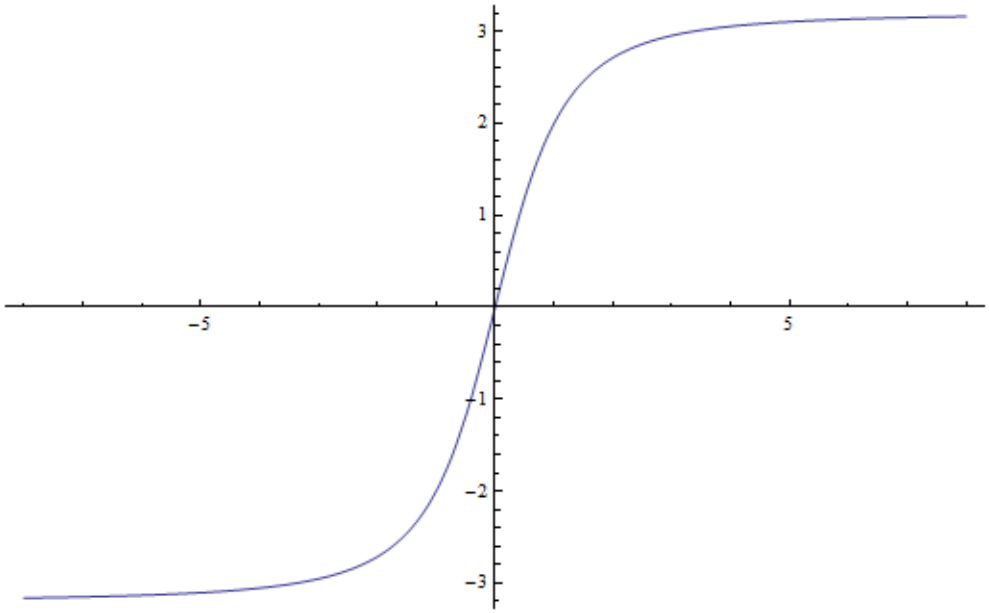
The first derivative is positive, so $G'(t)$ is a strictly increasing function. The second derivative is

$$G''(t) = \frac{-3rm_0^4b_1^2t}{(m_0^2t^2 + b_1^2)^{5/2}} \tag{17}$$

The second derivative is negative for $t > 0$, so the graph of $G(t)$ is concave for $t > 0$. The second derivative is positive for $t < 0$, so the graph of $G(t)$ is convex for $t < 0$. The origin $(0, 0)$ is an inflection point of the

graph. In fact, $G(t)$ is an odd function because $G(-t) = -G(t)$. The range of $G(t)$ is a bounded set because in the limiting sense, $G(+\infty) = rm_0$ and $G(-\infty) = -rm_0$. Figure 3 illustrates a typical graph.

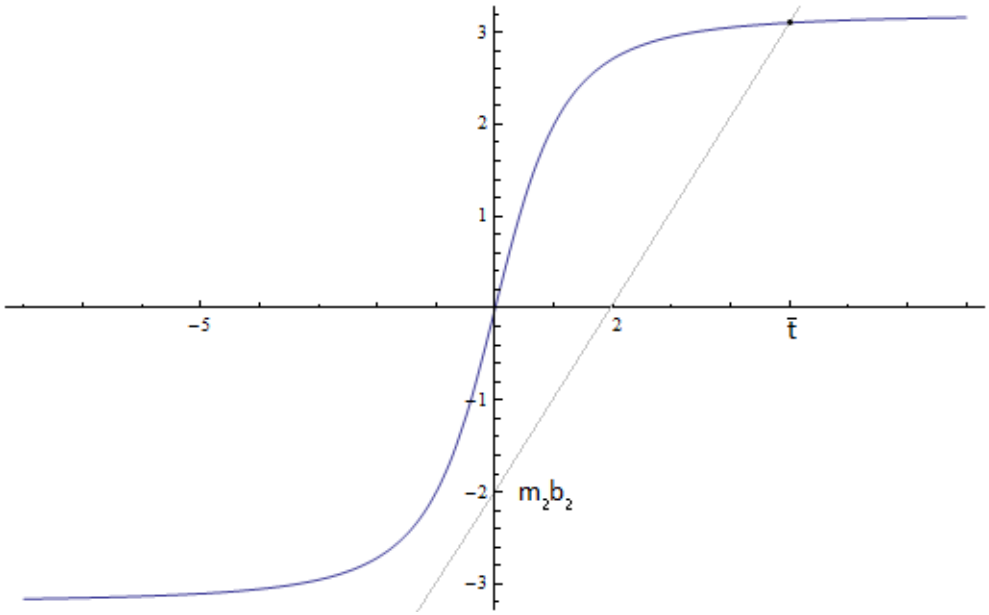
Figure 3. The graph for $G(t)$, where $\boldsymbol{N} = (0, 0, 1)$, $\boldsymbol{C} = (0, 0, 0)$, $\boldsymbol{M} = (4/5, 0, 3/5)$, $\boldsymbol{B} = (0, 1, 2)$, and $r = 4$. The asymptotes are at ± 3.2. The graph was generated by Mathematica [1].



Imagine overlaying the graph of $G(t)$ by a line $t + m_2 b_2$ that has slope 1 and intercept $-m_2 b_2$. The intersections of the line with the graph occur at the roots of $F'(t)$. For a root $\bar{t}$, the distance attains a local minimum when the slope of the graph at the intersection is smaller than 1 or attains a local maximum when the slope of the graph at the intersection is larger than 1. The conclusion is based on the second derivative test for $F(t)$ to classify the critical point: $F''(t) = 1 - G'(t)$. When $F'(t) = 0$ and $F''(t) > 0$, we have a local minimum. Similarly, $F''(t) < 0$ at the root implies we have a local maximum.

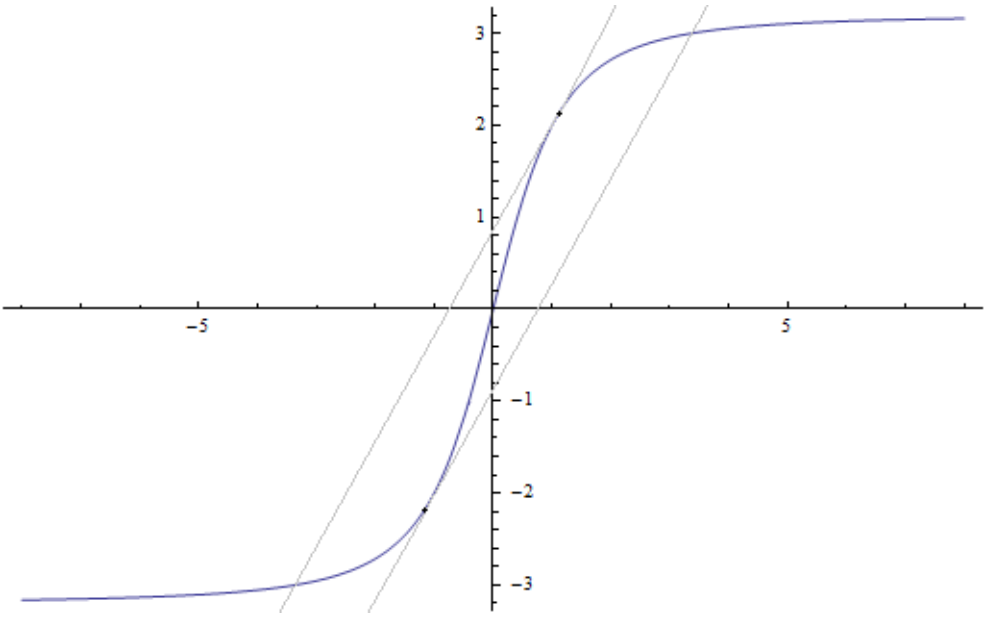
Because $G(t)$ is bounded as shown in figure 3, the line $t + m_2 b_2$ must intersect the graph in at least one point. We have exactly one point of intersection, call it $\bar{t} > 0$, when $m_2 b_2$ is negative with sufficiently large magnitude. Figure 4 illustrates.

Figure 4. The graph of $G(t)$ from figure 3 overlayed with a line that has a unique intersection $\bar{t}$ with the graph.



As you translate the line vertically upward, $\bar{t}$ must decrease. The line either intersects the graph only once *or* it becomes tangent to the graph at some value $-\hat{t} < 0$ and, necessarily, $G'(-\hat{t}) = 1$. Because $G''(t) > 0$ for $t < 0$, the derivative must increase for $t < 0$. Thus, the only way the line can become tangent to the graph at $-\hat{t} < 0$ is if $G'(0) > 1$. An algorithm for computing the intersections will know that there is a unique intersection when $G'(0) \leq 1$ *or* two or three intersections when $G'(0) > 1$. Figure 5 illustrates the tangency condition.

Figure 5. The graph of $G(t)$ from figure 3 overlayed with the tangent lines of slope 1. The tangents occur at $\pm\hat{t}$, where $\hat{t} \doteq 1.16684$. For the specific function, $m_2 b_2 = 6/5$, the line is above the tangents, and there is a unique intersection at $t \doteq -4.27118$.



For a line below or above the tangents, there is a unique intersection. The tangent lines have two intersections each. For a line between the tangents, there are three intersections. The middle intersection, however, corresponds to a local maximum in distance, so we may discard that from consideration. We may compute t-intervals that bound the roots that correspond to local minima. Bisection is used to generate the roots robustly.

First, we need to compute $G'(0)$ to decide whether we have one or two roots of interest. Using equation (16), $G'(0) = rm_0^2/|b_1|$. If $G'(0) \leq 1$, there is a unique root. If $m_2 b_2 = 0$, the root is clearly $\bar{t} = 0$. If $m_2 b_2 < 0$, the root must be positive. Moreover, it must lie between the intersection of the line with the t-axis and with the upper asymptote. The bounding interval is $[-m_2 b_2, -m_2 b_2 + rm_0]$. Similarly, if $m_2 b_2 > 0$, the root must be negative and is in the bounding interval $[-m_2 b_2 - rm_0, -m_2 b_2]$.

When $G'(0) = rm_0^2/|b_1| > 1$, the graph has tangent lines of slope 1 occurring at $\pm\hat{t}$. Using equation (16), solve $G'(\hat{t}) = 1$ for

$$\hat{t} = \left(\frac{\left(rm_0^2 b_1^2\right)^{2/3} - b_1^2}{m_0^2} \right)^{1/2} \tag{18}$$

The intercepts of the tangent lines with the G-axis are $\pm(G(\hat{t}) - \hat{t})$. The line is below the tangents when $m_2 b_2 < -(G(\hat{t}) - \hat{t})$, in which case there is a unique intersection. A bounding interval is $[-m_2 b_2, -m_2 b_2 + rm_0]$. Similarly, the line is above the tangents when $m_2 b_2 > (G(\hat{t}) - \hat{t})$, in which case there is a unique intersection. A bounding interval is $[-m_2 b_2 - rm_0, -m_2 b_2]$.

The line is strictly between the tangent lines when $-(G(\hat{t}) - \hat{t}) < m_2 b_2 < (G(\hat{t}) - \hat{t})$. When $m_2 b_2 \leq 0$, the largest root has a bounding interval $[-m_2 b_2, -m_2 b_2 + rm_0]$ and the smallest root has a bounding interval

$[-m_2b_2 - rm_0, -\hat{t}]$. When $m_2b_2 \geq 0$, the largest root has a bounding interval $[\hat{t}, -m_2b_2 + rm_0]$ and the smallest root has a bounding interval $[-m_2b_2 - rm_0, -m_2b_2]$. If the line is a tangent line, we know the root that generates the tangent line. The other root has bounding interval of the type described for the strict containment case. Once the roots are computed, we may compute the corresponding pairs of closest points and select that pair leading to minimum distance.

4.2.2 General Form

Line and Normal are Parallel

The point on the line closest to the circle is the intersection of the line with the plane of the circle, which happens to be the projection $\boldsymbol{Q}$ of $\boldsymbol{B}$ onto the plane. The closest point on the circle is then $\boldsymbol{K} = \boldsymbol{C} + r(\boldsymbol{Q} - \boldsymbol{C})/|\boldsymbol{Q} - \boldsymbol{C}|$ when $\boldsymbol{Q} \neq \boldsymbol{C}$; otherwise, all circle points are equidistant from the line.

Line and Normal are not Parallel

Equation (10) is similar to equation (14) except for the presence of the term $\boldsymbol{M} \times \boldsymbol{N} \cdot \boldsymbol{D} \times \boldsymbol{N}$. If zero, then we have the same form. If not zero, we can move the point $\boldsymbol{B}$ along the line to obtain a new point $\overline{\boldsymbol{B}} = \boldsymbol{B} + \lambda \boldsymbol{M}$ for which the two equations are of the same form. Define $\overline{\boldsymbol{D}} = \overline{\boldsymbol{B}} - \boldsymbol{C}$ and solve for λ in

$$\begin{aligned} 0 &= \boldsymbol{M} \times \boldsymbol{N} \cdot \overline{\boldsymbol{D}} \times \boldsymbol{N} \\ &= \boldsymbol{M} \times \boldsymbol{N} \cdot (\boldsymbol{D} + \lambda \boldsymbol{M}) \times \boldsymbol{N} \\ &= \boldsymbol{M} \times \boldsymbol{N} \cdot \boldsymbol{D} \times \boldsymbol{N} + \lambda |\boldsymbol{M} \times \boldsymbol{N}|^2 \end{aligned} \tag{19}$$

The solution is $\lambda = -(\boldsymbol{M} \times \boldsymbol{N} \cdot \boldsymbol{D} \times \boldsymbol{N})/|\boldsymbol{M} \times \boldsymbol{N}|^2$. Equation (10) becomes

$$F(t) = (t - \lambda) + \boldsymbol{M} \cdot \overline{\boldsymbol{D}} - r \frac{|\boldsymbol{M} \times \boldsymbol{N}|^2 (t - \lambda)}{\sqrt{|\boldsymbol{M} \times \boldsymbol{N}|^2 (t - \lambda)^2 + |\overline{\boldsymbol{D}} \times \boldsymbol{N}|^2}} \tag{20}$$

We may find roots of F' thought of as a function of $s = t - \lambda$ using the algorithm for the canonical form. The variable assignments are $m_0 = |\boldsymbol{M} \times \boldsymbol{N}|$, $m_2b_2 = \boldsymbol{M} \cdot \overline{\boldsymbol{D}}$, and $b_1^2 = |\overline{\boldsymbol{D}} \times \boldsymbol{N}|^2$.

An implementation is provided in the file DistLine3Circle3.h; see the function

```
Result Robust(Line3<Real> const& line, Circle3<Real> const& circle)
```

5 Distance Between Two Circles

Rename the first circle's parameters to $\boldsymbol{C}_0$, r_0, and $\boldsymbol{N}_0$. Let the second circle have center $\boldsymbol{C}_1$, radius r_1, and plane with normal $\boldsymbol{N}_1$. The second circle is parameterized by $\boldsymbol{P}(t) = \boldsymbol{C}_1 + r_1((\cos t)\boldsymbol{U}_1 + (\sin t)\boldsymbol{V}_1)$ for $t \in [0, 2\pi)$. The set $\{\boldsymbol{U}_1, \boldsymbol{V}_1, \boldsymbol{N}_1\}$ is right handed and orthonormal.

Define $\boldsymbol{D} = \boldsymbol{C}_1 - \boldsymbol{C}_0$ so that

$$\begin{aligned}
\boldsymbol{\Delta}(t) &= \boldsymbol{D} + r_1((\cos t)\boldsymbol{U}_1 + (\sin t)\boldsymbol{V}_1) \\
\boldsymbol{\Delta}'(t) &= r_1((-\sin t)\boldsymbol{U}_1 + (\cos t)\boldsymbol{V}_1) \\
\boldsymbol{N}_0 \times \boldsymbol{\Delta}(t) &= \boldsymbol{N}_0 \times \boldsymbol{D} + r_1((\cos t)\boldsymbol{N}_0 \times \boldsymbol{U}_1 + (\sin t)\boldsymbol{N}_0 \times \boldsymbol{V}_1) \\
\boldsymbol{N}_0 \times \boldsymbol{\Delta}'(t) &= r_1((-\sin t)\boldsymbol{N}_0 \times \boldsymbol{U}_1 + (\cos t)\boldsymbol{N}_0 \times \boldsymbol{V}_1)
\end{aligned} \tag{21}$$

Define $\gamma = \cos t$ and $\sigma = \sin t$. Equation (12) becomes

$$H(\gamma,\sigma) = (p_0(\gamma) + \sigma p_1(\gamma))(p_2(\gamma) + \sigma p_3(\gamma))^2 - r_0^2(p_4(\gamma) + \sigma p_5(\gamma))^2 = p_6(\gamma) + \sigma p_7(\gamma) \tag{22}$$

where the $p_i(\gamma)$ are polynomials in γ. Define $a_0 = r_1\boldsymbol{D}\cdot\boldsymbol{U}_1$, $a_1 = r_1\boldsymbol{D}\cdot\boldsymbol{V}_1$, $a_2 = |\boldsymbol{N}_0 \times \boldsymbol{D}|^2$, $a_3 = r_1\boldsymbol{N}_0 \times \boldsymbol{D}\cdot\boldsymbol{N}_0 \times \boldsymbol{U}_1$, $a_4 = r_1\boldsymbol{N}_0 \times \boldsymbol{D}\cdot\boldsymbol{N}_0 \times \boldsymbol{V}_1$, $a_5 = r_1^2|\boldsymbol{N}_0 \times \boldsymbol{U}_1|^2$, $a_6 = r_1^2\boldsymbol{N}_0 \times \boldsymbol{U}_1\cdot\boldsymbol{N}_0 \times \boldsymbol{V}_1$, and $a_7 = r_1^2|\boldsymbol{N}_0 \times \boldsymbol{V}_1|^2$; then

$$\begin{aligned}
p_0 &= (a_2 + a_7) + 2a_3\gamma + (a_5 - a_7)\gamma^2, & p_1 &= 2(a_4 + a_6\gamma) \\
p_2 &= a_1\gamma, & p_3 &= -a_0 \\
p_4 &= -a_6 + a_4\gamma + 2a_6\gamma^2, & p_5 &= -a_3 + (a_7 - a_5)\gamma
\end{aligned} \tag{23}$$

Using the multiplication rule $(q_0 + \sigma q_1)(q_2 + \sigma q_3) = (q_0q_2 + (1-\gamma^2)q_1q_3) + \sigma(q_0q_3 + q_1q_2)$ repeatedly, we obtain

$$\begin{aligned}
p_6 &= p_0(p_2^2 + (1-\gamma^2)p_3^2) + (1-\gamma^2)(2p_1p_2p_3) - r_0^2(p_4^2 + (1-\gamma^2)p_5^2) \\
p_7 &= 2p_0p_2p_3 + p_1(p_2^2 + (1-\gamma^2)p_3^2) - r_0^2(2p_4p_5)
\end{aligned} \tag{24}$$

where the degree of p_6 is at most 4 and the degree of p_7 is at most 3.

We need to eliminate σ in $0 = H(\gamma,\sigma) = p_6(\gamma) + \sigma p_7(\gamma)$. Solve for $p_6(\gamma) = -\sigma p_7(\gamma)$, square the terms, rearrange, and substitute $\sigma^2 = 1 - \gamma^2$ to obtain

$$\phi(\gamma) = p_6(\gamma)^2 - (1-\gamma^2)p_7(\gamma)^2 = 0 \tag{25}$$

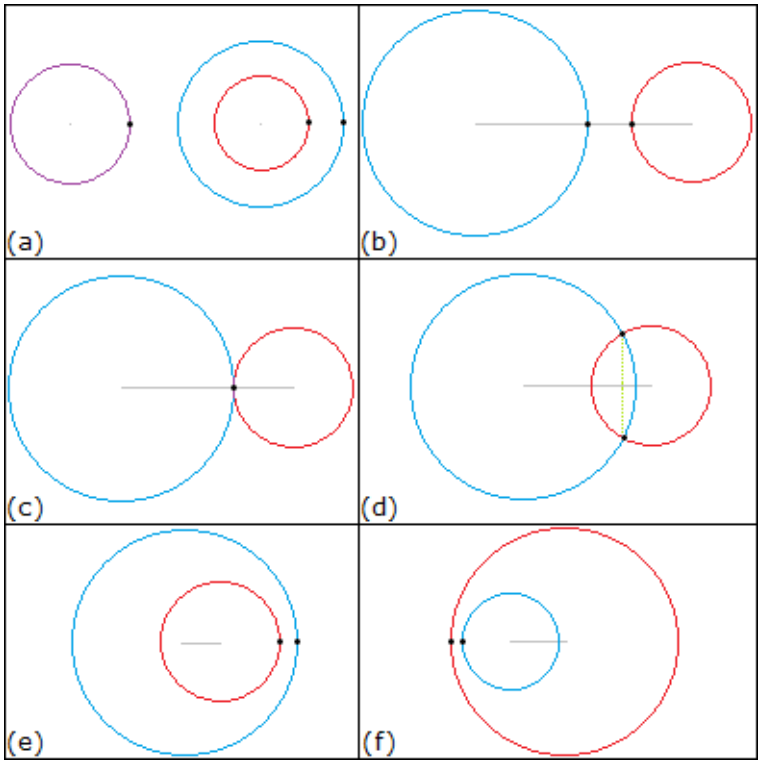
The polynomial $\phi(\gamma)$ has degree at most 8. Standard polynomial root finding algorithms can be used to compute the real-valued roots. For each root $\hat{\gamma}$, solve $H(\hat{\gamma},\hat{\sigma}) = 0$ for $\hat{\sigma} = -p_6(\hat{\gamma})/p_7(\hat{\gamma})$. Compute $\boldsymbol{P}(\hat{\gamma},\hat{\sigma})$, $\boldsymbol{K}(\hat{\gamma},\hat{\sigma})$, and $F(\hat{\gamma},\hat{\sigma})$. Keep track of the minimum F-value and the closest pair of points that generate it.

The previous paragraph presents the abstract concept, but as always with geometric algorithms, special cases arise that can be handled separately. The plane of the first circle has center $\boldsymbol{C}_0$ and is spanned by $\boldsymbol{U}_0$ and $\boldsymbol{V}_0$, where $\{\boldsymbol{U}_0, \boldsymbol{V}_0, \boldsymbol{N}_0\}$ is right handed and orthonormal.

5.1 Circles in the Same Plane

The second circle lives in the plane of the first circle when $\boldsymbol{N}_1 = \pm\boldsymbol{N}_0$ and $\boldsymbol{N}_0\cdot\boldsymbol{D} = 0$. Figure 6 shows the possibilities.

Figure 6. The first circle is drawn in blue and the second circle is drawn in red. The segment connecting centers is drawn in gray. The pairs of closest points are drawn in black. (a) Concentric circles. The two circles can be equal (drawn in violet) or not equal. (b) The circles are separated. (c) The circles are tangent with one circle outside the other. (d) The circles overlap and have two intersection points. (e) The first circle contains the second circle. (f) The second circle contains the first circle.



For equal circles, some point is selected as the common point for the pair of closest points and the distance is zero. For concentric and nonequal circles, a single pair of closest points is reported by the query, but a Boolean flag (equidistant) is set to true to indicate that there are infinitely many pairs. For separated circles (b), tangent circles (c), or containing circles, there is a single pair of closest points. Overlapping circles have two closest pairs.

The logic for determining which of the 6 cases you have is based on projecting the circles onto the line connecting the centers (if any). Cases (a) and (b) can be projected onto any line containing the common center. The origin of the line is chosen to be center $\boldsymbol{C}_0$. Relative to this origin, the other centers is $\boldsymbol{D} = \boldsymbol{C}_1 - \boldsymbol{C}_0$. The interval of projection for the first circle is $[-r_0, r_0]$ and the interval of projection for the second circle is $[d - r_1, d + r_1]$, where $d = |\boldsymbol{D}|$. Note that the classification is not symmetric in that the second circle has center always to the right of the first circle's center, where $\boldsymbol{D}$ points to the right (so to speak). This information is captured simply by $d \geq 0$.

The circles are separated (b) when $d > r_0 + r_1$ or tangent (c) when $d = r_0 + r_1$. The other cases occur when $d < r_0 + r_1$. If $d + r_1 \leq r_0$, the first circle contains the second circle (e). Note that $r_1 \leq d + r_1 \leq r_0$ implies $r_1 \leq r_0$. This condition and $d \geq 0$ force $[d - r_1, d + r_1] \subseteq [-r_0, r_1]$. If $d - r_1 \leq -r_0$, the second circle

contains the first circle (f); overlap cannot occur because we know $d \geq 0$. If both containment cases, we can distinguish (a) from (e) or (f) by testing whether or not d is zero.

Testing the cases in the order presented, the only remaining case is overlap of circles. The intersection points are shown in figure 6(d) on a vertical segment. The vertical line intersects the horizontal segment connecting the centers at a point $s\boldsymbol{D}$ for some value of $s \in (0,1)$. If $\boldsymbol{W} = \boldsymbol{D}/\boldsymbol{D}$, the vertical segment has direction $\boldsymbol{N}_0 \times \boldsymbol{W}$. The points of intersection of the circles are of the form $s\boldsymbol{D} + h\boldsymbol{N} \times \boldsymbol{W}$. Being on the first circle, we must have $|s\boldsymbol{D} + h\boldsymbol{N} \times \boldsymbol{U}|^2 = r_0^2$. Being on the second circle, we must have $|s\boldsymbol{D} - \boldsymbol{D} + h\boldsymbol{N} \times \boldsymbol{U}|^2 = r_1^2$. The equations have solution $s = (1 + (r_0^2 - r_1^2)/d^2)/2$ and $h = \pm\sqrt{r_0^2 - d^2 s^2}$.

An implementation is provided in the file DistCircle3Circle3.h; see the function

```
void DoQueryParallelPlanes(Circle3<Real> const& circle0, Circle3<Real> const& circle1,
    Vector3<Real> const& D, Result& result)
```

5.2 Circles in Parallel but Different Planes

This case is nearly identical to the same-plane case, except that $\boldsymbol{N}_0 \cdot \boldsymbol{D} \neq 0$. The closest point analysis is the same except that the points are 3-dimensional quantities. The function DoQueryParallelPlanes mentioned in the previous section also handles this case by working in the plane of the first circle but then adding the projection of $\boldsymbol{D}$ onto $\boldsymbol{N}_0$ to obtain the closest points.

5.3 Circles in Nonparallel Planes but Centers on Normal Line

The planes of the circle are not parallel when $\boldsymbol{N}_0 \times \boldsymbol{N}_1 \neq \boldsymbol{0}$. Let the center of the second circle be $\boldsymbol{C}_1 = \boldsymbol{C}_0 + h\boldsymbol{N}$ for some $h \neq 0$; that is, the centers are on the normal line of the first plane. To see how this causes the degree of $\phi(\gamma)$ to be smaller than 8, choose $\boldsymbol{V}_0 = \boldsymbol{N}_0 \times \boldsymbol{N}_1/|\boldsymbol{N}_0 \times \boldsymbol{N}_1|$ and $\boldsymbol{U}_0 = \boldsymbol{V}_0 \times \boldsymbol{N}_0$. We may then choose $\boldsymbol{V}_1 = \boldsymbol{V}_0$, because $\boldsymbol{V}_0$ is perpendicular to both $\boldsymbol{N}_0$ and $\boldsymbol{N}_1$ (it is a vector in both planes). Finally, $\boldsymbol{U}_1 = \boldsymbol{V}_1 \times \boldsymbol{N}_1$.

With the aforementioned choice of spanning vectors for the planes, some algebra will show that $a_1 = a_2 = a_3 = a_4 = a_6 = 0$, $a_0 \neq 0$, and $a_7 > a_5$. The polynomials are $p_0 = a_7 + (a_5 - a_7)\gamma^2$, $p_1 = 0$, $p_2 = 0$, $p_3 = -a_0$, $p_4 = 0$, and $p_5 = (a_7 - a_5)\gamma$. These lead to

$$p_6 = (\gamma^2 - 1)((a_7 - a_5)(a_0^2 + r_0^2(a_7 - a_5))x^2 - a_0^2 a_7), \quad p_7 = 0 \tag{26}$$

Two roots are always $\gamma = \pm 1$. The other quadratic factor has real-valued roots that—for r_0 sufficiently large—will be bounded by 1 in magnitude; however, these do not generate the closest pair of points. Geometrically, imagine starting with the second circle in a plane parallel to the first circle. Now tilt the second circle toward the first by rotating $\boldsymbol{U}_0$ to $\boldsymbol{U}_1$. The second circle's extreme point in the $\boldsymbol{U}_1$ direction closest to the first circle's plane will attain minimum distance. The rotation is about the line $\boldsymbol{C}_1 + s\boldsymbol{V}_0$, so it must be that $|\cos(t)| = 1$ and $\sin(t) = 0$.

5.4 General Case for Nonparallel Planes

In the previous section we had $p_7(\gamma)$ is identicallly zero. Generally, p_7 is not identically zero but it can be zero at a root of p_6. When solving for $\hat{\sigma} = -p_6(\hat{\gamma})/p_7(\hat{\gamma})$, we must test for $p_7(\hat{\gamma}) = 0$. For such roots $\hat{\gamma}$, we

examine the two choices $\hat{\sigma} = \pm\sqrt{1 - \hat{\gamma}^2}$.

An implementation is provided in the file DistCircle3Circle3.h; see the function

```
Result operator()(Circle3<Real> const&, Circle3<Real> const&)
```

The code tests whether $p_7(\gamma)$ is identically zero or not, having a separate block of code for each possibility. When $p_7(\gamma)$ is not identically zero, for each root $\hat{\gamma}$ of $\phi(\gamma)$ we test $p_7(\hat{\gamma})$ and branch accordingly. The result of all this logic is a set of pairs $(\hat{\gamma}, \hat{\sigma})$ that generate candidates for closest points. The minimum distance is tracked in this set, paying attention to whether there are two pairs of closest points and whether one circle point is equidistant from the other circle.

References

[1] Wolfram Research, Inc. *Mathematica 12.1.1.0*. Wolfram Research, Inc., Champaign, Illinois, 2020.